RECEIVED
CENTRAL FAX CENTER

☑001/027

MAR 0 7 2005

FAX TRANSMITTAL COVER SHEET

ORIGINAL WILL FOLLOW VIA:

CONLEY ROSE, P.C.
600 Travis, Suite 7100
Houston, Texas 77002
Fax Number: (713) 238-8008
Telephone Number: (713) 238-8000

- ☐ MAIL
- ☐ INTERNATIONAL AIRMAIL
- ☐ COURIER
- ☒ WILL NOT FOLLOW
- ☐ HAND DELIVERY
- ☐ WITH ENCLOSURE(S)
- ☐ WITHOUT ENCLOSURE(S)

PLEASE DELIVER THE FOLLOWING PAGES IMMEDIATELY TO:

NAME: **MAIL STOP APPEAL BRIEF - PATENTS**

FIRM: **U.S. PATENT AND TRADEMARK OFFICE**

CITY: **ALEXANDRIA, VIRGINIA**

FAX NO: **(703) 872-9306**

REMARKS: **Serial No. 10/037,806, filed 12/26/2001**
Attached hereto is an Appeal Brief for filing with the U.S. Patent and Trademark Office. Please acknowledge receipt of this facsimile.

Total Number of Pages (Including This One): **TWENTY-SEVEN (27)**

FROM: Mark E. Scott, Direct Dial No. (713) 238-8049

DATE: March 7, 2005

CLIENT/MATTER NO. 1662-49800 (200304304-1)

IF YOU DO NOT RECEIVE ALL THE PAGES,
PLEASE CALL BACK AS SOON AS POSSIBLE.

HEWLETT-PACKARD COMPANY
Intellectual Property Administration
P. O. Box 272400
Fort Collins, Colorado 80527-2400

COPY

PATENT APPLICATION

ATTORNEY DOCKET NO. 200304304-1

IN THE
UNITED STATES PATENT AND TRADEMARK OFFICE

Inventor(s):   Thomas J. BONOLA

Confirmation No.: 6235

Application No.: 10/037,806

Examiner: Midys Inoa

Filing Date: 12/26/2001

Group Art Unit: 2188

Title: METHOD FOR PROVIDING CONCURRENT NON-BLOCKING HEAP MEMORY MANAGEMENT FOR FIXED SIZED BLOCKS

Mail Stop Appeal Brief-Patents
Commissioner For Patents
PO Box 1450
Alexandria, VA 22313-1450

TRANSMITTAL OF APPEAL BRIEF

Sir:

Transmitted herewith is the Appeal Brief in this application with respect to the Notice of Appeal filed on __03/02/2005__ .

The fee for filing this Appeal Brief is (37 CFR 1.17(c)) $500.00.

(complete (a) or (b) as applicable)

The proceedings herein are for a patent application and the provisions of 37 CFR 1.136(a) apply.

( ) (a) Applicant petitions for an extension of time under 37 CFR 1.136 (fees: 37 CFR 1.17(a)-(d) for the total number of months checked below:

       ( ) one month $120.00
       ( ) two months $450.00
       ( ) three months $1020.00
       ( ) four months $1590.00

    ( ) The extension fee has already been filled in this application.

(X) (b) Applicant believes that no extension of time is required. However, this conditional petition is being made to provide for the possibility that applicant has inadvertently overlooked the need for a petition and fee for extension of time.

Please charge to Deposit Account 08-2025 the sum of __$500.00__ . At any time during the pendency of this application, please charge any fees required or credit any over payment to Deposit Account 08-2025 pursuant to 37 CFR 1.25. Additionally please charge any fees to Deposit Account 08-2025 under 37 CFR 1.16 through 1.21 inclusive, and any other sections in Title 37 of the Code of Federal Regulations that may regulate fees. A duplicate copy of this sheet is enclosed.

( ) I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Commissioner for Patents, Alexandria, VA 22313-1450. Date of Deposit:_____
              OR
(X) I hereby certify that this paper is being transmitted to the Patent and Trademark Office facsimile number (703) 872-9306 on 03/07/2005

Number of pages: 26

Typed Name: Colleen F. Brown

Signature: _____

Respectfully submitted,

Thomas J. BONOLA

By _____

Mark E. Scott
Attorney/Agent for Applicant(s)
Reg. No. 43,100

Date: 03/07/2005

Telephone No.: (713) 238-8000

BEST AVAILABLE COPY

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Appellant:	Thomas J. BONOLA	§	Confirmation No.:	6235	
		§			
Serial No.:	10/037,806	§	Group Art Unit:	2188	
		§			
Filed:	12/26/2001	§	Examiner:	Midys Inoa	
		§			
For:	Method For Providing	§	Docket No.:	200304304-1	
	Concurrent Non-Blocking	§			
	Heap Memory Management	§			
	For Fixed Sized Blocks	§			

APPEAL BRIEF

Mail Stop Appeal Brief – Patents   Date: March 7, 2005
Commissioner for Patents
PO Box 1450
Alexandria, VA 22313-1450

Sir:

Appellant hereby submits this Appeal Brief in connection with the above-identified application. A Notice of Appeal was filed via facsimile on March 2, 2005.

Appl. No. 10/037,806
Appeal Brief dated March 7, 2005
Reply to final Office action of January 4, 2005

TABLE OF CONTENTS

Appl. No. 10/037,806
Appeal Brief dated March 7, 2005
Reply to final Office action of January 4, 2005

I.   REAL PARTY IN INTEREST

The real party in interest is the Hewlett-Packard Development Company (HPDC), a Texas Limited Partnership, having its principal place of business in Houston, Texas, through its merger with Compaq Computer Corporation (CCC) which owned Compaq Information Technologies Group, L.P. (CITG). The assignment from the inventors to CITG was recorded on December 26, 2001, at Reel/Frame 012456/0350. The Change of Name document was recorded on May 12, 2004, at Reel/Frame 014628/0103.

Appl. No. 10/037,806
Appeal Brief dated March 7, 2005
Reply to final Office action of January 4, 2005

II. RELATED APPEALS AND INTERFERENCES

Appellant is unaware of any related appeals or interferences.

Appl. No. 10/037,806
Appeal Brief dated March 7, 2005
Reply to final Office action of January 4, 2005

III.   STATUS OF THE CLAIMS

Originally filed claims: 1-38.

Claim cancellations: 2 and 31.

Added claims: None.

Presently pending claims: 1, 3-30 and 32-38.

Presently appealed claims: 1, 3-30 and 32-38.

Appl. No. 10/037,806
Appeal Brief dated March 7, 2005
Reply to final Office action of January 4, 2005*

IV.   STATUS OF THE AMENDMENTS

No claims were amended after the final Office action dated
January 4, 2005.

145264.01/1662.49800   **Page 6 of 24**   HP PDNO 200304304-1

Appl. No. 10/037,806
Appeal Brief dated March 7, 2005
Reply to final Office action of January 4, 2005

V.   SUMMARY OF THE CLAIMED SUBJECT MATTER

The various embodiments of the invention are directed to a method of
providing concurrent non-blocking heap memory management for fixed size
blocks.[1] At least some of the illustrative embodiments are a method comprising
performing (by a software stream) heap memory operations on a first end of a
linked list of free heap memory of a heap pile,[2] and concurrently returning a return
block of heap memory (by a hardware device that used the return block of heap
memory) to the heap pile at a second end of the linked list of free heap memory.[3]

Other illustrative embodiments are a method of managing heap memory
comprising maintaining unused blocks of heap memory as a linked list (and
wherein the unused blocks of the linked list comprise a first block at a beginning
of the linked list, a second block pointed to the first block, and a third block at an
end of the linked list),[4] removing (by a software stream) the first block from the
linked list thus making the second block the beginning of the linked list,[5] and
returning a return block (by a hardware device that used the return block) to the
linked list by placing the return block at the end of the linked list.[6]

Yet other illustrative embodiments are a method of managing a heap
memory in a computer system comprising allowing a software thread to add and
remove blocks of heap memory from a linked list of free blocks of heap memory
in a last-in/first-out (LIFO) fashion at a first end of the linked list,[7] and allowing a
hardware device that uses blocks of heap memory to add the blocks of heap

---

[1] Specification Title.

[2] Specification Paragraph [0038], lines 1-8 within the paragraph; Paragraph [0042], lines 1-5;
Figures 5 and 6. Hereinafter, citations to the specification take the form [paragraph], lines [within
the paragraph] as a shorthand notation.

[3] [0045], lines 7-11; Figure 8.

[4] [0035], lines 1-18; Figures 2A and 2B.

[5] [0038], lines 1-8; [0042], lines 1-5; Figures 5 and 6.

[6] [0045], lines 7-11; Figure 8.

[7] [0038], lines 1-8; [0042], lines 1-5; Figures 5 and 6.

Appl. No. 10/037,806
Appeal Brief dated March 7, 2005
Reply to final Office action of January 4, 2005

memory to the linked list of free blocks of heap memory at a second end of the linked list.[8]

Yet other illustrative embodiments are computer system comprising a microprocessor executing a software stream,[9] and a main memory array (a portion of the main memory array allocated to be a heap memory, and wherein unused portions of the heap memory are part of a heap pile).[10] The heap pile further comprises a plurality of blocks where each block has a next block field, and where the heap pile is maintained as a linked list (each block's next block field pointing to a next block in the list).[11] The computer system further comprises a first bridge logic device coupling the microprocessor to the main memory array,[12] and a hardware device coupled to the heap memory through the first bridge logic device.[13] The software stream executed on the microprocessor removes blocks of heap memory from a beginning of the heap pile,[14] and simultaneously the hardware device returns blocks of heap memory used by the hardware device to an end of the heap pile.[15]

---

[8] [0045], lines 7-11; Figure 8.

[9] [0028], lines 2-7.

[10] [0029], lines 1-2; [0034], lines 1-2;

[11] [0035], lines 1-18; Figures 2A and 2B.

[12] [0028], lines 1-5; Figure 1.

[13] [0047], lines 3-6.

[14] [0038], lines 1-8; [0042], lines 1-5; Figures 5 and 6.

[15] [0045], lines 7-11; Figure 8.

Appl. No. 10/037,806
Appeal Brief dated March 7, 2005
Reply to final Office action of January 4, 2005

VI.   GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

Whether claims 12-28 are anticipated by Trainin (U.S. Pat. App.
No. 2002/0144073).

Whether claims 1, 3-11, 29-30 and 32-38 are unpatentable over Trainin in
view of Roohparvar (U.S. Pat. No. 6,504,768).

Appl. No. 10/037,806
Appeal Brief dated March 7, 2005
Reply to final Office action of January 4, 2005

VII. ARGUMENT

A.   Claims 1, 3-11, 29-30 and 32-38

Claims 1, 3-11, 29-30 and 32-38 stand rejected as allegedly obvious over Trainin in view of Roohparvar. Claim 1 is illustrative of this grouping of claims. This grouping is for purposes of this appeal only, and should not be construed to mean the patentability of any of the claims may be determined, in later actions before a court, based on the grouping. Rather, the presumption of 35 U.S.C. § 282 shall apply to each claim individually.

Trainin is directed to a method for memory heap management and buddy system management for service aware networks.[16] While Trainin may disclose free memory block management by way of a linked list,[17] Trainin makes no distinction between return of free memory blocks as between a software stream and a hardware device. Roohparvar is directed to redundancy selection in memory devices with concurrent read and write,[18] and is cited only for its concurrent aspects. It is noted, however, that Roohparvar deals with memory device level concurrent reads and writes, not management of free memory blocks, which could span several memory devices of the type disclosed in Roohparvar.

Claim 1, by contrast, specifically recites, "performing, **by a software stream**, heap memory operations on a first end of a linked list of free heap memory of a heap pile; and concurrently returning a return block of heap memory, **by a hardware device that used the return block of heap memory**, to the heap pile at a second end of the linked list of free heap memory." The Office Action dated January 4, 2005 takes the position that Trainin teaches differences in memory operations on a heap pile by a software stream and a hardware device citing Trainin's paragraph [0032].[19] This paragraph reads in full:

---

[16] Trainin Title.

[17] *See, e.g.,* Trainin Paragraph [0036].

[18] Roohparvar Title.

[19] Office action dated January 4, 2005, Page 8, first full paragraph.

> The present invention is described in the following exemplary embodiment with respect to a memory system that allocates memory blocks to tasks being executed using such memory. The present invention is also particularly well-suited for use in a SAN, where packet processors handle a plurality of process-flows, each with its own memory requirements. In SANs, process-flow processing is accomplished at wire speed, or the speed in which packets are moved through the network, which is paramount to the overall performance of the system.[20]

Applicant respectfully submits that the neither cited paragraph, nor Trainin in general, supports the proposition of differences in operation between software streams and hardware devices. Likewise, Trainin considered with Roohparvar fail to teach or fairly suggest the limitations of claim 1. For this reason alone, the rejection of this illustrative grouping of claims should be overturned.

The Office action further admits that Trainin fails to teach concurrently returning a block of heap memory to the heap pile while heap operations are being performed on the other end of the linked list.[21] In an attempt to fill this deficiency, the Office action relies on Roohparvar. However, Roohparvar does not supply the missing teaching regarding "returning a return block of heap memory, by a hardware device that used the return block of heap memory, to the heap pile at a second end of the linked list of free heap memory." Roohparvar is directed to **memory device level** concurrent reads and writes, not management of heap memory. The concurrent reads and writes **to a memory device** of Roohparvar, taken with Trainin, fail to teach or suggest "performing ... heap memory operations on a first end of a linked list **of free heap memory of a heap pile;** and **concurrently** returning a return block of heap memory ... to the heap pile at a second end of the linked list of free heap memory." For these additional reasons, the rejection of this illustrative grouping of claims should be overturned.

---

[20] Trainin Paragraph [0032].

[21] Office action dated January 4, 2005, Page 8, third full paragraph.

The Office action dated January 4, 2005, in the Response to Arguments section, states:

> [I]n allocating the memory blocks for use by a hardware device (see paragraphs 32-36), the system of Trainin is essentially using a software stream since hardware devices are driven by software.[22]

Even if hypothetically the statement regarding hardware devices being driven by software is true, Trainin and Roohparvar still fail to teach "concurrently returning a return block of heap memory, **by a hardware device that used the return block of heap memory,** to the heap pile at a second end of the linked list of free heap memory," whether that concurrent return is by way of a hardware state machine performing the necessary steps or software executing on the hardware device. In short, Trainin and Roohparvar fail to teach any distinction between the return of heap memory as between software and hardware, or that operations as between software and hardware's return of heap memory could take place concurrently.

Based on the foregoing, Appellant respectfully submits that the rejections of the claims in this first grouping be reversed, and the grouping set for issue.

B.   Claims 12, 14-26 and 28

Claims 12, 14-26 and 28 stand rejected as allegedly anticipated by Trainin. Claim 12 is illustrative of this grouping of claims. This grouping is for purposes of this appeal only, and should not be construed to mean the patentability of any of the claims may be determined, in later actions before a court, based on the grouping. Rather, the presumption of 35 U.S.C. § 282 shall apply to each claim individually.

Trainin is directed to a method for memory heap management and buddy system management for service aware networks.[23]   While Trainin may disclose free memory block management by way of a linked list,[24] Trainin makes no distinction between return of free memory blocks as between a software stream and a hardware device.

---

[22] Office action dated January 4, 2005, Page 13, fourth full paragraph.

[23] Trainin Title.

[24] *See, e.g.,* Trainin Paragraph [0036].

Claim 12, by contrast, specifically recites, "maintaining unused blocks of heap memory as a linked list, and wherein the unused blocks of the linked list comprise a first block at a beginning of the linked list, a second block pointed to the first block, and a third block at an end of the linked list; **removing, by a software stream, the first block from the linked list**, thus making the second block the beginning of the linked list; and **returning a return block, by a hardware device that used the return block, to the linked list by placing the return block at the end of the linked list**." The advantage of such a system is that the return and removal of memory blocks may take place simultaneously.[25] The Office action dated January 4, 2004 takes the position that Trainin distinguishes between memory operations on a heap pile by a software stream and a hardware device citing Trainin's paragraph [0032].[26] However, the cited paragraph, or Trainin in general, fails to support this proposition. Trainin fails to teach any distinction between the return of heap memory as between software and a hardware device that use the heap memory, or the advantage of such a system presents in the form of simultaneous operations.

Based on the foregoing, Appellant respectfully submits that the rejections of the claims in this second grouping be reversed, and the grouping set for issue.

C. Claims 3, 13 and 27

Claims 3, 13 and 27 stand rejected as either anticipated by Trainin (claims 13 and 27) or obvious over Trainin and Roohparvar (claim 3). Claim 13 is illustrative of this grouping of claims. This grouping is for purposes of this appeal only, and should not be construed to mean the patentability of any of the claims may be determined, in later actions before a court, based on the grouping. Rather, the presumption of 35 U.S.C. § 282 shall apply to each claim individually.

Training discloses a "root table 400" that, for each category of block sizes, identifies the first free block of that size. Trainin's Figure 4 is reproduced below for convenience of the discussion.

---

[25] [0048], lines 1-3.

[26] Office action dated January 4, 2005, Page 8, first full paragraph.

Appl. No. 10/037,806
Appeal Brief dated March 7, 2005
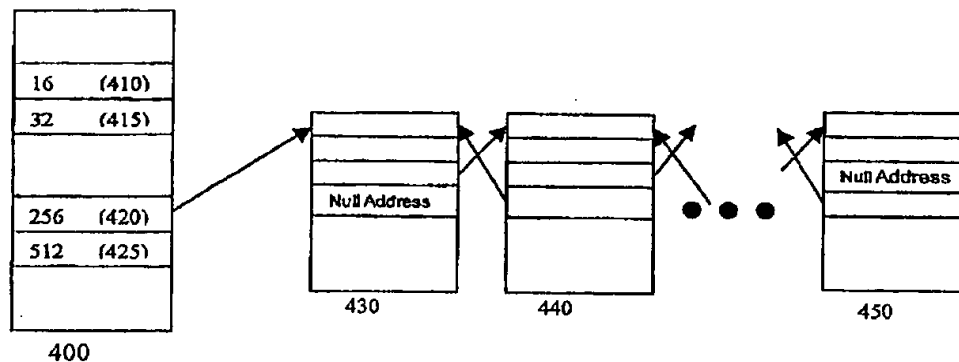Reply to final Office action of January 4, 2005

Fig. 4

With regard to this figure, Trainin states:

> Management of free blocks is done through a linked list of free blocks, each block having a free block header, as illustrated with reference to FIG. 4. A list of the permitted sizes of the free blocks, from the smallest to the largest, that may be allocated in the system is used as the root table 400. Each entry corresponds to a block size, for example 16 bytes 410, 32 bytes 415, 256 bytes 420, 256 bytes 425 and so on. **Each such entry contains an address that points to the header of the first free block 430 of the size managed by that entry.** If there is no free block corresponding to that size, a "null address" value is used.[27]

Thus, Trainin teaches only an entry or pointer to the first free block in the linked list.

Claim 13, by contrast, specifically recites, "reading a bottom register, the bottom register identifying the third block; writing a block number of the return block to a next state field of the third block; and writing the block number of the return block to the bottom register." As defined in claim 12, from which claim 13 depends, the third block is the last block in the linked list, and thus the "bottom

---

[27] Trainin Paragraph [0036] (emphasis added).

register" defined points to the end of the linked list.[28] Trainin, as discussed immediately above, teaches only a pointer to the **first block** of the linked list in the "root table 400." Thus, Trainin does not teach or suggest "reading a bottom register, the bottom register identifying the third block... and writing the block number of the return block to the bottom register." In fact, it appears that in Trainin the only way to find the last block of the linked list is to step through each block's "next free block address 330"[29] until the block with a null entry is found.

Applicants therefore respectfully submit that claim 3 is not rendered obvious by Trainin and Roohparvar, and that claims 13 and 27 are not anticipated by Trainin.

Based on the foregoing, Appellant respectfully submits that the rejections of the claims in this third grouping be reversed, and the grouping set for issue.

---

[28] *C.f.* claim 14 that defines a top register.

[29] Trainin Figure 3.
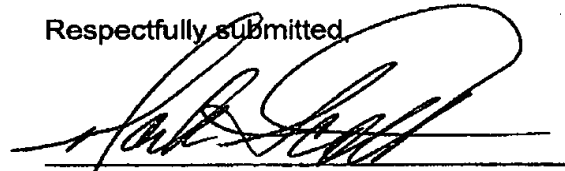
Appl. No. 10/037,806
Appeal Brief dated March 7, 2005
Reply to final Office action of January 4, 2005

VIII. CONCLUSION

For the reasons stated above, Appellant respectfully submits that the Examiner erred in rejecting all pending claims. It is believed that no extensions of time or fees are required, beyond those that may otherwise be provided for in documents accompanying this paper. However, in the event that additional extensions of time are necessary to allow consideration of this paper, such extensions are hereby petitioned under 37 C.F.R. § 1.136(a), and any fees required (including fees for net addition of claims) are hereby authorized to be charged to Hewlett-Packard Development Company's Deposit Account No. 08-2025.

Respectfully submitted,

Mark E. Scott
PTO Reg. No. 43,100
CONLEY ROSE, P.C.
(713) 238-8000 (Phone)
(713) 238-8008 (Fax)
ATTORNEY FOR APPELLANT

HEWLETT-PACKARD COMPANY
Intellectual Property Administration
Legal Dept., M/S 35
P.O. Box 272400
Fort Collins, CO 80527-2400

BEST AVAILABLE COPY

Appl. No. 10/037,806
Appeal Brief dated March 7, 2005
Reply to final Office action of January 4, 2005

IX.   CLAIMS APPENDIX

1.   (Previously presented) A method comprising:

performing, by a software stream, heap memory operations on a first end
        of a linked list of free heap memory of a heap pile; and concurrently

returning a return block of heap memory, by a hardware device that used
        the return block of heap memory, to the heap pile at a second end
        of the linked list of free heap memory.

2.   (Cancelled).

3.   (Previously presented) The method as defined in claim 1 wherein
returning a return block of heap memory further comprises:

        writing a null to a next block field of the return block of heap memory;

        writing a block number of the return block of heap memory to a next block
                field of a last block of heap memory in the linked list;

        changing the contents of a bottom register to point to the return block of
                heap memory; and thereby

        making the return block of heap memory a last entry in the linked list.

4.   (Previously presented) The method as defined in claim 1 wherein
performing heap memory operations further comprises returning, by the software
stream, a second block of heap memory by placing the second block of heap
memory at the first end of the linked list.

5.   (Previously presented) The method as defined in claim 4 wherein
returning the second block of heap memory at the first end of the linked list by the
software stream further comprises:

        determining a block number of a primary block of heap memory resident at
                the first end of the linked list;

        writing the block number of the primary block of heap memory to a next
                block field of the second block; and

writing atomically a block number of the second block to a top register...

6. (Previously presented) The method as defined in claim 5 wherein determining a block number of a primary block of heap memory resident at the first end of the linked list further comprises reading the top register prior to writing the block number of the second block.

7. (Previously presented) The method as defined in claim 1 wherein performing heap memory operations further comprises removing, by the software stream, heap memory from the first end of the linked list.

8. (Previously presented) The method as defined in claim 7 wherein removing heap memory from the linked list heap management system further comprises taking a primary block of heap memory resident at the first end of the of the linked list.

9. (Previously presented) The method as defined in claim 8 wherein taking a primary block of heap memory further comprises:
    determining a block number of the primary block;
    reading a next block field of the primary block of memory; and
    removing the primary block if the next block field of the primary block does
        not indicate a null.

10. (Original) The method as defined in claim 9 wherein determining a block number of the primary block further comprises reading a top register, wherein the top register identifies the beginning of the linked list.

11. (Original) The method as defined in claim 9 wherein removing the primary block if the next block field of the primary block does not indicate a null further comprises writing a block number of the next block field of the primary block to the top register.

12.   (Previously presented) A method of managing a heap memory comprising:

maintaining unused blocks of heap memory as a linked list, and wherein
the unused blocks of the linked list comprise a first block at a
beginning of the linked list, a second block pointed to the first block,
and a third block at an end of the linked list;

removing, by a software stream, the first block from the linked list, thus
making the second block the beginning of the linked list; and

returning a return block, by a hardware device that used the return block,
to the linked list by placing the return block at the end of the linked
list.

13.   (Previously presented) The method of managing a heap memory as
defined in claim 12 wherein returning a return block further comprises:

writing a null to a next block field of the return block;

reading a bottom register, the bottom register identifying the third block;

writing a block number of the return block to a next state field of the third
block; and

writing the block number of the return block to the bottom register.

14.   (Original) The method of managing a heap memory as defined in claim 12
wherein removing, by a software stream, the first block from the linked list further
comprises:

reading a top register, the top register identifying the first block;

reading a next block field of the first block, the next block field of the first
block identifying the second block; and

writing a block number of the second block to the top register.

15.   (Original) The method of managing a heap memory as defined in claim 14
wherein writing a block number of the second block to the top register further

comprises atomically writing the block number of the second block to the top and minimum register.

16. (Original) The method of managing a heap memory as defined in claim 12 further comprising returning, by a software stream, a fourth block to the linked list by placing the fourth block at the beginning of the linked list, thus making the fourth block the beginning of the linked list.

17. (Original) The method of managing a heap of memory as defined in claim 16 wherein returning a fourth block to the linked list by placing the fourth block at the beginning of the linked list further comprises:

> reading a top register, the top register identifying the beginning of the linked list;

> writing a block number of the block identified by the top register to a next state field of the fourth block; and

> writing a block number of the fourth block to the top register.

18. (Original) The method of managing a heap memory as defined in claim 17 wherein writing a block number of the fourth block to the top register further comprises atomically writing the block number of the fourth block to the top register.

19. (Previously presented) A method of managing a heap memory in a computer system, the method comprising:

> allowing a software thread to add and remove blocks of heap memory from a linked list of free blocks of heap memory in a last-in/first-out (LIFO) fashion at a first end of the linked list; and

> allowing a hardware device that uses blocks of heap memory to add the blocks of heap memory to the linked list of free blocks of heap memory at a second end of the linked list.

20.   (Original) The method of managing a heap memory in a computer system as defined in claim 19 wherein allowing a software thread to remove blocks of heap memory in LIFO fashion further comprises:

      determining, by the software thread, a block number of a block of heap memory at the first end of the linked list; and

      removing the block of heap memory at the first end of the linked list.

21.   (Original) The method of managing a heap memory in a computer system as defined in claim 20 determining a block number of a block of heap memory at the first end of the linked list further comprises reading a beginning register that stores a block number of a block of heap memory at the first end of the linked list.

22.   (Original) The method of managing a heap memory in a computer system as defined in claim 21 wherein removing the block of heap memory at the first end of the linked list further comprises:

      reading a next block field of the block of heap memory at the first end of the linked list to identify a block number of a next block in the linked list; and

      writing the block number of the next block in the linked list to the beginning register.

23.   (Original) The method of managing a heap memory in a computer system as defined in claim 20 wherein allowing a software thread to add blocks of heap memory in LIFO fashion further comprises:

      determining, by the software thread, a block number of a block of heap memory at the first end of the linked list;

      writing the block number of the block of heap memory at the first end of the linked list to a next block field of a return block of heap memory; and

      making the return block of heap memory the first end of the linked list.

24.   (Original) The method of managing a heap memory in a computer system as defined in claim 23 wherein determining a block number of a block of heap memory at first end of the linked list further comprises reading a beginning register that stores a block number of a block of heap memory at the first end of the linked list.

25.   (Original) The method of managing a heap memory in a computer system as defined in claim 24 wherein making the return block of heap memory the first end of the linked list further comprises writing a block number the return block of heap memory to the beginning register.

26.   (Previously presented) The method of managing a heap memory in a computer system as defined in claim 20 wherein allowing a hardware device that uses blocks of heap memory to add the blocks of heap memory to the linked list of free blocks of heap memory at a second end of the linked list further comprises:

determining, by the hardware device, a block number of a block of heap memory at the second end of the linked list;

writing, by the hardware device, a block number of a return block of heap memory to a next block field of the block of heap memory at the second end of the linked list; and

making the return block of heap memory the second end of the linked list.

27.   (Original) The method of managing a heap memory in a computer system as defined in claim 26 wherein determining a block number of a block of heap memory at the second end of the linked list further comprises reading an end register that stores a block number of the block of heap memory at the second end of the linked list.

28.   (Original) The method of managing a heap memory in a computer system as defined in claim 27 wherein making the return block of heap memory the

Appl. No. 10/037,806
Appeal Brief dated March 7, 2005
Reply to final Office action of January 4, 2005

second end of the linked list further comprises writing a block number the return block of heap memory to the end register.

29.   (Previously presented) A computer system comprising:

a microprocessor executing a software stream;

a main memory array, a portion of the main memory array allocated to be a heap memory, and wherein unused portions of the heap memory are part of a heap pile, the heap pile further comprising

a plurality of blocks;

each block having a next block field; and

wherein the heap pile is maintained as a linked list, each block's next block field pointing to a next block in the list;

a first bridge logic device coupling the microprocessor to the main memory array;

a hardware device coupled to the heap memory through the first bridge logic device;

wherein the software stream executed on the microprocessor removes blocks of heap memory from a beginning of the heap pile; and simultaneously

the hardware device returns blocks of heap memory used by the hardware device to an end of the heap pile.

30.   (Previously presented) The computer system as defined in claim 29 wherein the plurality of blocks each have the same number of bytes.

31.   (Cancelled).

32.   (Previously presented) The computer system as defined in claim 29 further comprising the software stream returns blocks to the heap pile at the

beginning of the heap pile simultaneously as the hardware device returns blocks of heap memory used by the hardware device to the end of the heap pile.

33.   (Original) The computer system as defined in claim 29 wherein the hardware device is the graphics card.

34.   (Original) The computer system as defined in claim 29 wherein the hardware device is a network interface card.

35.   (Original) The computer system as defined in claim 29 wherein the hardware device is an audio card.

36.   (Original) The computer system as defined in claim 29 wherein the hardware device is a mass storage device.

37.   (Original) The computer system as defined in claim 36 wherein the mass storage device is a hard drive.

38.   (Original) The computer system as defined in claim 37 wherein the mass storage device is compact disk storage device.

145264.01/1662.49800   **Page 24 of 24**   HP PDNO 200304304-1

PAGE 27/27 * RCVD AT 3/7/2005 4:31:56 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-1/4 * DNIS:8729306 * CSID:7132388008 * DURATION (mm-ss):11-02